基于可达矩阵的软件体系结构演化波及效应分析*

王映辉[1,2+], 张世琨[1], 刘 瑜[3], 王立福[1]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](陕西师范大学 计算机学院,陕西 西安 710062)

[3](北京大学 信息遥感与地理信息系统研究所,北京 100871)

Ripple-Effect Analysis of Software Architecture Evolution Based on Reachability Matrix

WANG Ying-Hui[1,2+], ZHANG Shi-Kun[1], LIU Yu[3], WANG Li-Fu[1]

[1](Institute of Software, Peking University, Beijing 100871, China)

[2](School of Computer Science, Shanxi Normal University, Xi'an 710062, China)

[3](Institute of Remote Sensing and Geographic Information Systems, Peking University, Beijing 100871, China)

+ Corresponding author: Phn: +86-10-62767117 ext 102, E-mail: wyh_925@163.com, http://www.pku.edu.cn

Received 2004-01-16; Accepted 2004-03-29

Wang YH, Zhang SK, Liu Y, Wang LF. Ripple-Effect analysis of software architecture evolution based on reachability matrix. *Journal of Software*, 2004,15(8):1107~1115.

http://www.jos.org.cn/1000-9825/15/1107.htm

Abstract: Construction and evolution are two basic properties of software. Software evolution consists of a series of complex change activities. Software complexity decides that the research of software evolution should start with the macroscopical level firstly. Software architecture (SA), which acts as a blueprint and a skeleton of software, offers an availability approach with the whole macroscopical software architecture and evolution grasped by people. The component, connector models, which create SA relation matrix and reachability matrix, are described. Depending on matrix shift and calculation, ripple-effect of SA evolution can be analyzed and its quantity can be ascertained, describing every ripple-effect caused by component deletion, addition, modification, division and combination respectively. At the same time, an approach for calculating the relative quantity of component effect is described. All are credible foundation for management, control, usage and evaluation of SA evolution, and are foundation for SA evolution automation calculation based on matrix shift in computer.

Key words: software architecture (SA); evolution; interactive relationship; SA (software architecture) reachability matrix; ripple-effect

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA113171 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312006 (国家重点基础研究发展规划(973)); the National Postdoctoral Research Foundation of China under Grant No.20040350251(国家博士后基金)

作者简介: **王映辉**(1967－),男,甘肃庄浪人,博士,副教授,主要研究领域为软件工程,软件演化;**张世琨**(1969－),男,博士,副教授,主要研究领域为软件工程,软件体系结构;**刘瑜**(1970－),男,博士,副教授,主要研究领域为软件工程,GIS;**王立福**(1945－),男,博士,教授,博士生导师,主要研究领域为软件工程.

摘 要: 构造性和演化性是软件的两个基本特性.软件演化由一系列复杂的变化活动组成.软件演化的复杂性决定了对软件演化的研究首先应从宏观层面入手.软件体系结构 SA 作为软件的蓝图和支撑骨架,为人们宏观把握软件的整体结构和软件演化提供了一条有效的途径.描述了 SA 的构件——连接件模型,建立了 SA 关系矩阵和可达矩阵,凭借矩阵变换与运算对 SA 演化中的波及效应进行了深入的分析和量化界定;并对演化中的构件删除、增加和修改以及构件的合并与分解等变化活动所引起的各种波及效应给予了阐述;同时,给出了构件在 SA 中贡献大小相对量的计算方法.为 SA 演化的管理、控制、利用和评价提供了可靠的依据,并为基于矩阵变换的 SA 演化的计算机自动处理奠定了基础.

关键词: 软件体系结构;演化;交互关系;软件体系结构可达矩阵;波及效应

中图法分类号: TP311 **文献标识码**: A

构造性和演化性是软件的两个基本特性[1].软件进行渐变并达到所希望的形态就是软件演化.软件演化由一系列复杂的变化活动组成.变化是软件中存在的普遍现象.对软件变化的控制是软件开发者历来追求的目标.引起软件变化的原因是多方面的,如基础设施的改变、功能需求的增加、高性能算法的发现、技术环境因素的变化等等.所以,对软件变化甚至演化进行理解和控制显得比较复杂而又困难.

一方面,软件演化的复杂性决定了软件演化研究首先应从宏观层面入手,这样可以避免过早地陷入软件演化研究的复杂细节中;另一方面,对软件演化的控制首先要求我们能够确定每一个变化活动所影响的范围.

近几年,软件体系结构 SA(software architecture)已成为软件研究的热点之一.它作为软件的蓝图,为人们宏观把握软件的整体结构提供了一条有效途径.SA 的形式化描述、基于构件和 SA 的软件组装与开发、SA 实践、面向模式的 SA 研究等等,这一切已构成了 SA 的基本技术和理论体系.显而易见,要从宏观角度来刻画软件演化,并对演化中的局部效应进行观察和控制,自然应从 SA 演化研究开始.另外,SA 是软件生命周期的早期产品,着重解决软件系统的结构和需求向实现平坦过渡的问题,是软件生命周期中开发、集成、测试和更改阶段的基础,加之对 SA 检测和修改的相对低代价性[2],所以深入研究 SA 的演化是非常必要的.

目前,在对 SA 的演化研究方面,形式化描述方法[3,4]更加偏重于对软件构造性的描述,而将 SA 的演化性隐藏在各种"运算"之中,不能直观地反映 SA 演化活动的波及效应.UML 方法[5]对 SA 的描述基于不同的视图,虽然从不同的角度实现了对 SA 结构的可视性,但并未涉及 SA 演化活动中波及效应的确定方法.其他有关软件演化的研究基本注重软件演化过程的定性分析[6~8],而以 SA 为核心进行软件演化定量分析的文章并不多见.

本文第 1 节描述 SA 的构件——连接件模型.第 2 节建立 SA 模型下的关系矩阵和可达矩阵,并给出转换方法.第 3 节着重对基于关系矩阵和可达矩阵的 SA 演化波及效应进行刻画,提出波及效应范围的界定和构件对 SA 贡献的相对量大小的确定方法.第 4 节对目前的研究进行分析和说明.第 5 节是本文的结论.

1 软件体系结构模型及其概念

目前对 SA 的定义形式多样.为了研究方便,本文采用许多文献中公认的[2,6,9,10]简单定义,即 SA 是组成系统的构件以及构件与构件之间交互作用关系(连接件)的高层抽象.

1.1 构 件

构件是软件系统的构成要素和结构单元,是软件功能设计、实现和寄居状态的承载体.所以,从系统的构成上看,任何在系统运行中承担一定功能、发挥一定作用的软件体都可看成构件.

1.1.1 构件模型

本文将构件看作"黑盒"而不追求构件的内部结构.但必须定义构件的统一形态,即构件模型(如图 1 所示).

构件模型是对构件本质特征的抽象描述,使关心和使用构件的外部环境(如使用构件构造出的应用系统、构件组装辅助工具和构件复用者等)能够在一致的概念模型下观察和使用构件[2,9].

构件内部包含有内部规约(specification),主要是语法约束、语义模型和其他一些服务特性等[9]以及自己的内部体系结构.从表示形式上看,构件是具有操作接口定义的抽象数据类型描述.它由内部数据结构及其结构上

的操作和对外提供的操作方法集合组成.操作接口是构件对外提供的操作根据不同的应用目的所划分成的子集合.

在不同的开发和应用环境中,构件模型可以找到它所对应的概念,如模块、控件、库、设计模式、体系结构模式、框架等.

1.1.2 构件接口

接口是构件之间进行交互作用的通道,也是唯一的交流途径.它代表了构件在不同环境下的交互内容.由此可见,构件通过接口定义了与外界信息的传递和承担的系统责任,除此之外,环境不应对构件作任何与接口无关的假设,如实现细节等.

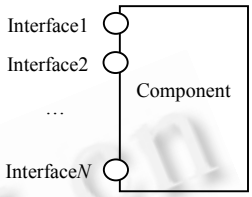


Fig.1 Component model

图 1 构件模型

接口根据不同的目的和环境可以设计和规约为不同的表现形式.如,可采用与平台无关和高抽象性的标准接口描述语言 IDL(interface description language)来定义.又如,在基于层次消息总线的软件体系结构风格 JB/HMB 中[11],基于事件消息机制,将接口定义为发出和接收的消息集合,在增强了构件复用潜力的同时,提高构件之间连接的灵活性,便于软件系统的演化.

构件的接口可分为两类[9],构件供外部使用的接口即功能规约(function specification)以及构件用到的外部接口即接入点(entry point),本文并不严格区分.

1.2 连接件

任何构件的独立存在并不能发挥和实现自己的功能.连接件是构件之间联系的特殊部件,实现了构件间信息交换和行为关系.本文中所指的连接件蕴涵着:

(1) 表示构件之间的交互关系;

(2) 承载着构件之间的交互语义;

(3) 拥有构件及其之间关系的约束;

(4) 是由构件形成的拓扑结构的承担者.

连接件模型如图 2 所示.它也拥有其内部结构和外部接口.但是,构件可以在多个交互中扮演不同的角色,而连接件只能在一个交互中起作用,这种不同在接口上表现为:构件可以有若干个相互无关的接口,而连接件只能有一组相互关联的接口[2,9].

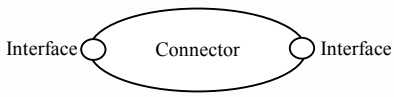


Fig.2 Connector model

图 2 连接件模型

连接件的连接本质是由连接的实现机制和信息交换协议(连接的规约)决定的.简单的连接只有实现机制,构件间最简单的连接(如操作功能的调用等)可以通过直接联系完成(连接件只蕴涵着方向语义);复杂连接由实现机制和协议复合而成,而这种复合由连接件来承担并完成(此种意义上的连接件也可以称为构件).如果将连接件看成特殊类型的构件,它与普通意义上的构件的差别主要是在构成系统或 SA 时的作用不同.一般构件是软件功能的承载部件,而连接件是负责完成构件间信息交换和行为的专用部件.可见,连接件的功能是实现构件之间的行为连动或转动以及信息交换.

连接件具有连接的方向性和连接的角色性.前者是指连接件的任何一端可以进行单向或双向请求传递,后者是指参与连接一方的作用和地位,有主动和被动或请求和响应之分.

从 SA 演化的角度来看,只有一端(接口)服务连接在系统中的连接件是无意义的.

1.3 SA模型

尽管对 SA 还没有形成统一的认识,但有一点可以肯定的是,它是对系统的高层次抽象,应从粗粒度部件(构件和连接件)的角度来把握系统层的结构和行为.为此,在构件和连接件模型的基础上容易建立 SA 模型.

假设一个系统的 SA 由 5 个构件和 6 个连接件构成,其交互关系模型如图 3 所示.其中,构件 Component1 通过连接件 Connector1,Connector3,Connector5 和 Connector6 分别与构件 Component2,Component3,Component4

和 Component5 发生交互关系,其他构件之间的关系亦然.

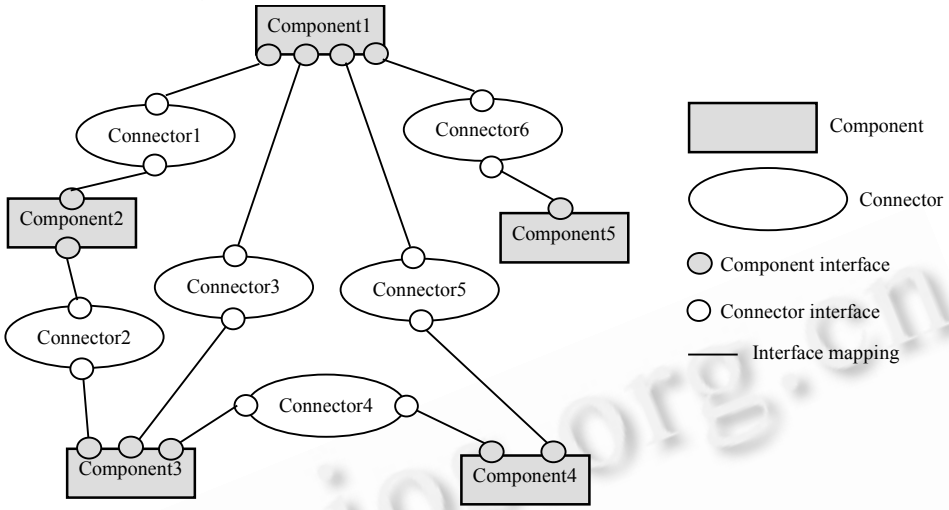


Fig.3 Sample of SA model
图 3 SA 模型示例

如果保留图 3 中连接件的基本语义即方向语义,则得到如图 4 所示的有向图.其中,构件 Component2 和构件 Component3 之间存在双向连接语义,说明图 3 中的连接件 Connector2 至少承载着双向的交互方向语义关系.

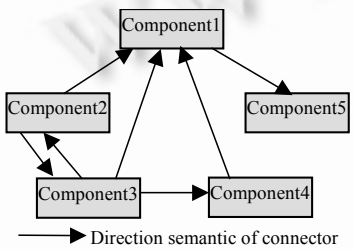


Fig.4 Predigestion model of Fig.3
图 4 图 3 的简化模型

该SA模型的一个重要特性是将交互看成了一个建模的概念.直观地看,SA 简化模型图中的线对应的就是连接件.虽然图 4 是一种非常简化的 SA 模型,但它在宏观层面上为本文研究 SA 的演化波动效应分析提供了足够的信息.

由此可见,SA 可形式化地描述为 $SA=\{components,connectors\}$,其中,*components* 是构件(component)的有限集,*connectors* 是连接件(connector)的有限集.而构件间的交互关系是通过连接件来体现的.

2 SA 关系矩阵与变换

图 4 虽然由图 3 简化而来,但对本文在宏观层面研究 SA 演化中的波动效应分析非常有参考价值.也就是说,前者使 SA 结构(structure)更为清晰,有利于 SA 结构变化(如构件的删除、增加等)所引起的 SA 静态演化的研究;而后者则保留了更为丰富的交互语义(通过连接件的显式存在),有利于 SA 动态演化的深入研究.

2.1 SA结构关系邻接矩阵图

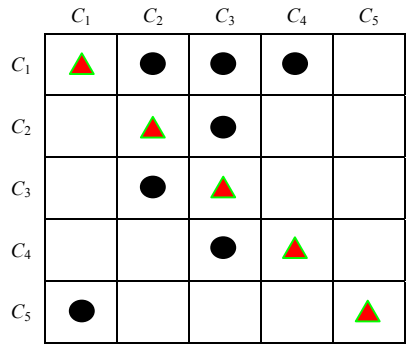
图 5 是由图 4 构造出的 SA 结构关系邻接矩阵图.图 5 中每一个黑色的圆圈关联着两个构件(在此指具有直接交互方向关系的两个构件),其方向性由行和列分别表示,行为被关联的末尾构件,列为被关联的起始构件.如,第 1 行第 2 列的圆圈表示构件 C_1(Component1)与构件 C_2(Component2)存在直接交互关系,并自构件 C_2 指向构件 C_1.

另外,图 5 中的三角形表示构件的自交互关系,这种关系隐藏在构件自身内部.由于构件自身是有体系结构的,即其内部一定存在交互关系,所以在图 5 中用有别于不同构件间交互关系的三角形标出.本文并不关心构件自身内部的交互关系.

2.2 SA语义关系连接矩阵图

图 6 是由图 3 构造出的 SA 语义关系连接矩阵图.图中三角形的含义与图 5 中的相同,黑色方块表示行列中

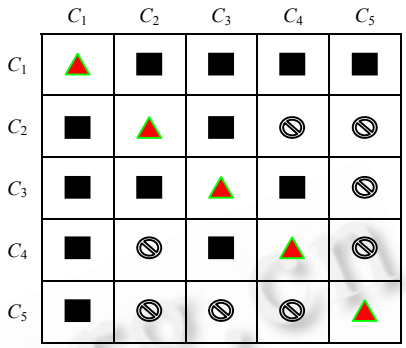
对应两个构件之间存在着直接的交互语义关系,带横线的圆圈表示行列中对应两个构件之间不存在或存在间接的交互语义关系.



C_1:Component1, C_2:Component2, …, C_5:Component5

Fig.5 SA Structure relationship adjacency matrix graph of Fig.4

图 5 图 4 的 SA 结构关系邻接矩阵图



C_1:Component1, C_2:Component2, …, C_5:Component5

Fig.6 SA semantic relationship connective matrix graph of Fig.3

图 6 图 3 的 SA 语义关系连接矩阵图

图 6 中的黑色方块和带横线的圆圈是对不同环境和应用中语义形式的抽象,在具体化时,矩阵图在表面上的这种对称性可能会被打破,并变为非对称形式(非对称矩阵图).例如,当连接件的语义表示两个构件间最大无环可达路径上构件的个数(包括两端)时,列 C_3 和行 C_5 交叉处的圆圈变为数值 4(如图 4 所示),即 $C_3 \rightarrow C_2 \rightarrow C_1 \rightarrow C_5$,而列 C_5 与行 C_3 交叉处的圆圈变为数值 0,即不可达.

由此可见,SA 结构关系邻接矩阵图是 SA 语义关系连接矩阵图的一种简单形式.也就是说,在 SA 结构关系邻接矩阵中,我们将语义信息简化成了构件间是否存在直接交互以及交互的方向语义,有利于 SA 结构演化中波及效应的研究.

2.3 SA关系矩阵及其变换

给 SA 结构关系邻接矩阵图和 SA 语义关系连接矩阵图中的十字交叉处填充上具有一定语义的数值以后,所形成的对应矩阵分别称为 SA 结构关系邻接矩阵和 SA 语义关系连接矩阵.SA 结构关系邻接矩阵简称 SA 关系矩阵或关系矩阵.

例如,在图 4 中,当自构件 C_i 到构件 $C_j(i,j=1,2,\ldots,5)$存在一条直接的连接时,图 5 中的第 C_i 行和第 C_j 列交叉处填充为 1,否则为 0,其他连接依此类推.这样,就形成了对应的 SA 结构关系邻接矩阵.

又如,在图 3 中,当自构件 C_i 到构件 $C_j(i,j=1,2,\ldots,5)$存在一条可达路径时,将图 6 中的第 C_i 行和第 C_j 列交叉处填充为 1,否则为 0,其他连接依此类推.这样,就形成了对应的 SA 语义关系连接矩阵.

由图论的相关理论[12]可知,SA结构关系邻接矩阵通过一定的运算(如闭包运算)后可以转换为SA语义关系连接矩阵.

3 SA 演化中的波及效应分析及其构件贡献

由于系统需求、技术、环境和分布等因素的变化而最终导致的 SA 按照一定的目标形态的变动,称为 SA 演化.按照演化时间,SA 演化可划分为静态演化和动态演化两个方面.对 SA 在非运行时刻的修改和变更称为 SA 的静态演化(如软件版本的升级等),而软件在运行时刻的 SA 变换称为 SA 的动态演化.按照演化的内容,SA 的演化又可分为结构演化和语义演化.Darwin 和 C_2 都直接支持 SA 的动态结构演化,CHAM,Wright,Rapide 支持 SA 的动态语义演化[6].C_2 中则通过专门的 SA 修改语言 AML 来实现 SA 的动态性,Darwin 则采用脚本语言对 SA 进行修改;CHAM 对 SA 的变换采用多值演算的方法来实现,而 Wright 则通过顺序通信进程代数(CSP)来描述构件的交互语义.如前所述,这些形式化的方法对 SA 演化的支持具有相当的隐蔽性,不利于使用.为此,本文提出了基于SA模型的关系矩阵和可达矩阵(定义见下)支持下SA演化的波及效应量化分析,同时给出了确定构件

对 SA 贡献大小的确定方法.

3.1 SA静态演化中的波及效应和构件贡献

SA 的静态演化是指在系统非运行状态下,软件功能的变更和环境因素的变化等对组成 SA 的构件进行的增加、替换、删除、重新组合和拆分等操作所引起的 SA 的变化.在 SA 的静态演化中,表面上看是对构件的增加、替换、删除,但这种变化蕴涵着一系列的连带和波及效应,更多地表现为变化的构件或连接件与其相关联的构件或连接件的重新组合和归整.

3.1.1 构件波及范围的界定和贡献

设 SA 的关系矩阵 $M_R=(C_{ij})$,其中 C_{ij} 表示构件 C_i 与构件 C_j 之间的连接关系;$i,j=1,2,\ldots,n$,并且

$$C_{ij}=\begin{cases}1, & \text{当}C_i\text{与}C_j\text{之间存在直接交互关系时}\\ 0, & \text{当}C_i\text{与}C_j\text{之间不存在直接交互关系时}\end{cases}.$$

另外,设集合 $X=\{C_1,C_2,\ldots,C_i,\ldots,C_n\}$,则关系矩阵 M_R 对应的关系: $R\subseteq X^2$,于是,关系 R 的传递闭包 $R^+=R\cup R^2\cup\ldots\cup R^n$,则 R^+ 对应的矩阵 $M_{R^+}=M_R\vee M_{R^2}^2\vee\ldots\vee M_{R^n}^n=\bigvee_{k=1}^{n}M_{R^k}^k$,其中 $M_{R_i}^i=M_{R_{i-1}}^{i-1}\vee M_R$, $i=2,3,\ldots,n$.此时,称 M_{R^+} 为 SA 的可达矩阵,简称可达矩阵.

例如,如图 5 所示的 SA 对应的关系矩阵 $M_R=\begin{bmatrix}0&1&1&1&0\\0&0&1&0&0\\0&1&0&0&0\\0&0&1&0&0\\1&0&0&0&0\end{bmatrix}$,则 SA 的可达矩阵 $M_{R^+}=\begin{bmatrix}0&1&1&1&0\\0&1&1&0&0\\0&1&1&0&0\\0&1&1&0&0\\1&1&1&1&0\end{bmatrix}$.

由可达矩阵各列可以直接得出(如图 4 所示):构件 C_1(第 1 列)到构件 C_5(第 5 行)可达;构件 C_2(第 2 列)和 C_3(第 3 列)到其他构件(各行)分别都可达;构件 C_4(第 4 列)到 C_1(第 1 行)和 C_5(第 5 行)分别可达;构件 C_5(第 5 列)到其他构件都不可达.

由此可见,通过可达矩阵非常容易界定某一构件变化所影响的其他构件.进而,当某一组构件发生变化时,可以圈定被影响或波及到的其他构件的范围.

另外,通过可达矩阵,可以对各个构件对 SA 影响的大小(称为贡献或贡献大小)进行确定和排序.具体办法是,求取可达矩阵各列的元素之和,其大小代表相应列头构件所影响的其他构件的个数.当然,这个数值越大,该构件对 SA 的影响(贡献)越大.例如,图 4 的可达矩阵 M_{R+}的各列头构件 C_1,C_2,C_3,C_4 和 C_5 对应的各列元素之和分别为 1,5,5,2 和 0,所以各构件对 SA 贡献的大小排列为 $C_2\geq C_3>C_4>C_1>C_5$.

3.1.2 SA 静态演化中的基本活动与波及影响讨论

SA 静态演化中的基本活动包括删除构件、增加构件、修改构件、构件合并与构件分解这 5 种,具体分析如下:

(1) 删除 SA 中一个构件 C_i

如果可达矩阵的第 i 列全为 0,则可以直接删除构件 C_i,而不会对其他构件造成影响(但会影响 SA 的结构).例如,删除图 4 中的构件 Component5 不会影响其他构件,具体表现可能为软件功能的删减.需要注意的是,当删除第 i 列构件 C_i 时,引起第 i 行构件 C_i 的删除也不影响其他构件,因为构件在可达矩阵行中的含义仅仅是被影响者.

如果可达矩阵的第 i 列至少存在一个 1,则删除此构件 C_i 对 SA 的结构和整个软件的功能及性能等会发生影响,其影响的相对大小为第 i 列元素之和.如删除构件 C_4(对应的列元素之和为 2)没有删除构件 C_3(对应的列元素之和为 5)对 SA 的影响大(同时参照图 4).

(2) 在 SA 中增加一个构件 C_i

当增加一个构件时,首先要提供与之有直接交互关系的构件,进而形成新的 SA 可达矩阵,最后在 SA 范围内判定被影响的全部构件.其影响的相对大小是新的 SA 可达矩阵的第 i 列元素之和.

(3) 在 SA 中修改一个构件 C_i

构件的修改包括对构件自身结构的调整和功能的增减,可能会导致 SA 在结构上或语义上的变化.这种变化应能由实施构件变更的人员所把握.也就是说,这种变化直接影响的构件是之前被确定的.因而,新的 SA 与其对应的可达矩阵也能被唯一确定.

(4) 对 SA 中的一簇构件实施合并

对一簇构件进行合并的操作是经常发生的,如功能的合并、小构件组装成更大的构件、遗留构件的进一步包装(符合新的标准)等都可引起构件的合并.

在构件的合并过程中,一是以关系矩阵和可达矩阵为依据,指导构件的有效合并;二是在被合并的构件簇确定的前提下,新的 SA 可达矩阵的求取.

对第 1 种情况,可通过对可达矩阵进行分块划分和合并的办法来实现,但行和列分别合并后形成的新构件要一致.如行中第 2 和第 3 构件合并,一定保证在列中第 2 和第 3 构件的合并;此外,新块的形成以 0 或 1 居多者为原则;另外,合并时采用布尔和运算.

例如,对上例中的可达矩阵 M_{R^+} 分块划分为

$$\left[\begin{array}{c|cc|cc} 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right],$$

且合并成如下的矩阵:

$$M_{\text{merge}}=\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

则由此可得新的 SA,如图 7 所示.

在图 7 中,构件 Component23 是构件 Component2 和 Component3 合并后的新构件,构件 Component45 是构件 Component4 和 Component5 合并后的新构件;而虚线箭头表示构件 Component23 和构件 Component45 自己具有内部可达性(符合图 4 的情况).将图 7 与图 4 相对比,其意义是显然的.

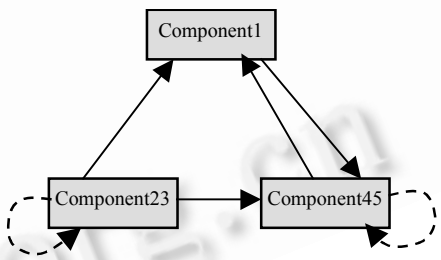


Fig.7 New SA model after emerging

图 7 合并后新的 SA 模型

对于第 2 种情况,由构件的合并可以直接得出新的 SA 可达矩阵.当然,构件的合并遵循“并”运算.由于直接关系矩阵是完备的(完全表达了组成 SA 的全部构件之间的邻接方向关系),所以不会由于构件的合并而造成新的构件间额外交互关系的出现.

(5) 将 SA 中的某一个构件 C_i 拆分成若干个构件 $C_{i1},C_{i2},\ldots,C_{im}$

构件拆分为构件合并的逆过程,新的构件间交互关系和新的 SA 可达矩阵的确定比较简单,不再重述.

最后要说明的一点是,在对 SA 的静态演化分析中,我们几乎全部忽略了构件间的语义(仅仅保留了方向语义),但可以看出,这对于宏观层面研究 SA 的静态演化的波及效应分析是非常必要的.

3.2 SA动态演化中的波及效应

SA 的动态演化比静态演化更为复杂.在对 SA 的静态演化分析中,主要集中在对 SA 静态结构的波及影响范围的分析和界定上,几乎全部忽略了交互的语义信息(仅仅保留了方向语义),这对于 SA 静态演化分析是可行的.但是,在进行 SA 的动态演化分析时,对 SA 结构中构件的删除、增加、修改以及构件的合并和分解等活动相对甚少(即使有,也可用静态演化分析方法),也就是说,基本保持了 SA 在结构上的稳定性,而变化主要集中在系统运行时语义的“波及”上.为了宏观地把握这种动态性,我们假定被研究系统的 SA 静态结构是不变的,且构成了稳定的网状通道,在系统运行时,语义信息经过这个网状通道传播,即语义信息的范围随着运行状态的变化而扩张或收缩,这种扩张或收缩反映了系统运行在某一时刻 SA 的形态.也就是说,系统运行到某一时刻(状态),在

静态 SA 中可以圈定相应的传播范围,此范围内被涉及到的构件和连接件重新构成了此时此刻新的 SA.

当给连接件赋予更丰富的语义后,对 SA 的动态演化研究才有意义.也就是说,SA 的动态演化分析不是只通过关系矩阵和可达矩阵就能简单描述的,但以 SA 静态演化描述中的关系矩阵和可达矩阵为基础,结合相关理论(如润湿理论、元胞自动机理论、马尔科夫理论或语义网的一些研究成果等),也能对 SA 的动态演化进行有效的分析.我们在这方面的研究已经获得了一定的进展,鉴于篇幅有限,将另文加以阐述.

4 相关研究

综观软件演化研究的历史,早在 1978 年,Yau 就将模块变化的影响在软件模块之间的传播称为"变化传播(change propagation)"[13].从 20 世纪 90 年代初期开始,随着软件数量的增加,特别是基于 Internet 的分布式大规模软件系统的进一步应用,研究者将软件变化分析提到了比较关注的位置.如 Bohner 在其论文[7,14]中引用入"波及效应"一词来形象地描述软件变换的影响,并用可达矩阵的概念对软件变化进行了简单的阐述,但没有给出组成软件的要素(相当于构件)对软件贡献大小的概念,更没有进行 SA 演化基本活动的波及效应分析.20 世纪 90 年代中期,人们对软件生命周期中软件变化影响的两个关键阶段(需求和维护),进行了软件变化捕捉和实施方法的研究.其中代表性的有,Baxter 等人提出了通过将设计信息延伸到维护过程中来掌握软件的变化[15],Chiang 提出了软件转换的增量式快速原型,并用其来形式化用户需求和规约,进而构造和管理软件的演化[16],但都偏向于定性的过程分析.20 世纪 90 年代中后期,软件变化影响分析研究在面向对象的软件系统中相对较多,如 Ryder 等人提出的多态探讨影响分析[8],Chaumun 等人提出的类层次影响计算方法[17].这些方法都建立在软件原子成分的划分及成分间关系的确定上,并进而来分析变化的影响,但对影响的类型、途径、程度和范围几乎未加考虑.Zhou 在她的博士学位论文中较好地解决了这些问题[18].以上这些面向对象软件变化的分析方法主要集中在较细微的实现层次上.目前,对软件变化的研究日益增多,如 Erich 等人提出了软件变化的可视化方法[19],Zeng 提出通过智能工作流技术来获取软件的易变性[20],Lai 提出了行为关注(behavioural concerns)的软件变化建模方法来实现对软件变化建模[21].

5 结束语

我们基于 SA 的构件-连接件模型,建立了 SA 关系矩阵和可达矩阵,凭借矩阵变换和运算直接对 SA 演化,特别是 SA 的静态演化中的波及效应,进行了深入的刻画分析和量化界定,并对静态演化中的构件的删除、增加和修改以及构件的合并和分解等变化活动(演化的表现形态)所引起的各种波及效应进行了区分和阐述;提出了构件贡献的概念,并给出了其大小相对量的计算方法.这一切为 SA 演化的管理、控制、利用和评价提供了可靠的依据,并为基于矩阵变换的 SA 演化的计算机自动处理奠定了基础.

进一步的研究工作包括对 SA 动态演化的深入刻画、SA 演化中的波及效应能量衰减计算和界定、基于 SA 可达矩阵的 SA 演化过程评估、计算机辅助 SA 演化工具的研制和应用等.

致谢 非常感谢北京大学信息学院软件研究所软件演化讨论班上的老师和同学们的热烈讨论.

References:

[1] Liu Y, Zhang SK, Wang LF, Yang FQ. Component-Based software frameworks and role extension form. Journal of Software, 2003,14(8):1364~1370 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1364.htm

[2] Bass L, Clements PC, Kazman R. Software Architecture in Practice. Aonton: Addison-Wesley, 1998.

[3] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description languages. IEEE Trans. on Software Engineering, 2000,26(1):70~93.

[4] Luo HJ, Tang ZS, Zheng JD. Visual Architecture Description Language XYZ/ADL. Journal of Software, 2000,11(8):1024~1029 (in Chinese with English Abstract).

[5] Rational Rose Corporation. UML notation guide. 2003. http://www.rational.com/uml

[6] Sun CA, Jin MZ, Liu C. Overviews on software architecture research. Journal of Software, 2002,13(7):1228~1237 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/1228.pdf

[7] Bohner SA. Impact analysis in the software change process: A year 2000 perspective. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM'96). Washington: IEEE, 1996. 42~51.

[8] Ryder BG, Tip F. Change impact analysis for object-oriented programs. In: Proc. of 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. New York: ACM Press, 2001. 46~53.

[9] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. Journal of Software, 2003:14(4):721~732 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/721.htm

[10] Garlan D, Shaw M. An introduction to software architecture. In: Ambriola V, Tortora G, eds. Advances in Software Engineering and Knowledge Engineering, Vol II. Hackensack: World Scientific Publishing, Co., 1993.

[11] Zhang SK, Wang LF, Yang FQ. Software architecture style based tier message bus. Science in China (Series E), 2002,32(3): 393~400 (in Chinese with English abstract).

[12] Li PL, Li LS, Li Y, Wang CL. Discrete Mathematics. Beijing: Higher Education Press, 2001 (in Chinese).

[13] Yau SS, Collofello JS, McGregor TM. Ripple effect analysis of software maintenance. In: Proc. of the Computer Software and Applications Conf. (COMPSAC'78). Piscataway: IEEE Computer Society Press, 1978. 60~65.

[14] Bohner SA. Software change impacts: An evolving perspective. In: Proc. of the Int'l Conf. of Software Maintenance (ICSM 2002). Washington: IEEE, 2002. 263~272.

[15] Baxter ID, Pidgeon CW. Software change through design maintenance. In: Proc. of the Int'l Conf. of Software Maintenance. Washington: IEEE, 1997. 250~259.

[16] Chiang CC, Urban JE. Incremental elicitation and formalization of user requirements through rapid prototyping via software transformations. In: Proc. of the 20th Int'l Computer Software and Applications Conf. (COMPSAC'96). Washington: IEEE, 1996. 240~245.

[17] Chaumun MA, Kabaili H, Keller RK, Lustman F. A change impact model for changeablity assessment in object-oriented software systems. In: Proc. of the 3rd European Conf. on Software Maintenance and Reengineering. Washington: IEEE, 1999. 130~138.

[18] Zhou X. Research on object-oriented software change impact analysis technology [Ph.D. Thesis]. Beijing: Peking University, 2003 (in Chinese with English abstract).

[19] Erich SG, Graves TL, Karr AF, Mockus A, Schuster P. Visualizing software changes. IEEE Trans. on Software Engineering, 2002,28(4):396~412.

[20] Zeng DD, Zhao JL. Achieving software flexibility via intelligent workflow techniques. In: Proc. of the 35th Annual Hawaii Int'l Conf. on System Sciences (HICSS-35 2002). Washington: IEEE, 2002. 606~615.

[21] Lai AY, Murphy GC. Behavioural concern modeling for software change tasks. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM 2002). Washington: IEEE, 2002. 112~121.

附中文参考文献:

[1] 刘瑜,张世琨,王立福,杨芙清.基于构件的软件框架与角色扩展形态研究.软件学报,2003,14(8):1364~1370. http://www.jos.org.cn/1000-9825/14/1364.htm

[4] 骆华俊,唐雅松,郑建丹.可视化体系结构描述语言 XYZ/ADL.软件学报,2000,11(8):1024~1029.

[6] 孙昌爱,金茂忠,刘超.软件体系结构研究综述.软件学报,2002,13(7):1228~1237. http://www.jos.org.cn/1000-9825/13/1228.pdf

[9] 梅红,陈锋,冯耀东,杨杰.ABC:基于体系结构、面向构件的软件开发方法.软件学报,2003,14(4):721~732. http://www.jos.org.cn/1000-9825/14/721.htm

[11] 张世琨,王立福,杨芙清.基于层次消息总线的软件体系结构风格.中国科学(E 辑),2002,32(3):393~400.

[12] 李盘林,李丽双,李洋,王春立.离散数学.北京:高等教育出版社,2001.

[18] 周欣.面向对象软件变化影响分析技术研究[博士学位论文].北京:北京大学,2003.